EXP05 - Statistical Inference

September 4, 2023

Exp_No: 05

Reg No.: URK21CS1128

Bewin Felix R A

Aim:

To demonstrate the statistical interferences used for data science application using python language.

Description:

Inferential statistics are used to draw inferences from the sample of a huge data set. Random samples of data are taken from a population, which are then used to describe and make inferences and predictions about the population.

Sample Mean and Population Mean:

Sample mean is the arithmetic mean of random sample values drawn from the population. Population mean represents the actual mean of the whole population. If the sample is random and sample size is large then the sample mean would be a good estimate of the population mean.

Correlation Coefficient:

The correlation coefficient quantifies the relationship between the two variables. There are two methods of calculating the Correlation Coefficient and its matrix – Pearson and Spearman.

Covariance Matrix:

It is a square matrix giving the covariance between each pair of elements of a given random vector.

Hypothesis Testing using Z Test:

Hypothesis testing is a statistical method that is used in making statistical decisions using experimental data. One of the ways to perform hypothesis testing is Z-test, where the Two-sample Z-test is used to test whether the two datasets are similar or not. Also, Z-test is used when the sample size is greater than 30.

Confidence Interval:

A confidence interval displays the probability that a parameter will fall between a pair of values around the mean. Confidence intervals measure the degree of uncertainty or certainty in a sampling method. They are most often constructed using confidence levels of 95% or 99%.

[13]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

```
import numpy as np
import scipy.stats as stats
import math
```

[34]:
```
print('URK21CS1128')
path = "supermarket.csv"
df = pd.read_csv(path) #read the csv file
print(df.shape) #3000 - population
```

```
URK21CS1128
(3000, 17)
```

[15]:
```
# Lets take seed so that everytime the random values come out to be constant
np.random.seed(6)
```

[16]:
```
#1. Calculate the sample mean for ‘Unit price’ column with n=500 and observe
print('URK21CS1128')
s1 = 500
sample_1 = np.random.choice(a = df['Unit price'] , size = s1)
m_sample_1 = sample_1.mean();
print("The Mean of the sample data of Unit price for 500 :", m_sample_1)
```

```
URK21CS1128
The Mean of the sample data of Unit price for 500 : 55.11994
```

[17]:
```
#2. Calculate the sample mean for ‘Unit price’ column with n=1000 and observe
print('URK21CS1128')
s2 = 1000

sample_2 = np.random.choice(a = df['Unit price'] , size = s2)
m_sample_2 = sample_2.mean();
print("The Mean of the sample data of Unit price for 1000 :", m_sample_2 )
```

```
URK21CS1128
The Mean of the sample data of Unit price for 1000 : 56.03247999999999
```

[18]:
```
#3. Calculate the population mean for ‘Unit price’ column
print('URK21CS1128')
pm = df['Unit price'].mean()
print("The Mean of the population data of Unit price :", pm)
```

```
URK21CS1128
The Mean of the population data of Unit price : 55.67213
```

[19]:
```
#4. Calculate the confidence interval (CI) with sample mean for ‘Unit price’
 ↪column of n=500 and confidence level of 95%. Observe whether the population
 ↪mean lies in CI.
print('URK21CS1128')
print( "Sample mean of 500 samples:", m_sample_1)
```

```
SD = sample_1.std()
print("Sample SD of 500 samples:", SD)

CL=0.95
alpha=(1-CL)/2
z_critical = round(stats.norm.ppf(1-alpha),2)
print("Z-score:", z_critical)

er=z_critical*(SD/math.sqrt(s1))
L=m_sample_1-er
H=m_sample_1+er

print("Confidence Level", L, H)
print("[",L,pm,H,"]")
```

```
URK21CS1128
Sample mean of 500 samples: 55.11994
Sample SD of 500 samples: 26.150292078605926
Z-score: 1.96
Confidence Level 52.827765835805906 57.412114164194094
[ 52.827765835805906 55.67213 57.412114164194094 ]
```

[20]:

```
#5. Change the confidence level to 99% and observe the confidence interval for␣
↪the same sample mean for 'Unit price' column of n=500.
print('URK21CS1128')
print( "Sample mean of 500 samples:", m_sample_1)

SD = sample_1.std()
print("Sample SD of 500 samples:", SD)

CL=0.9
alpha=(1-CL)/2
z_critical = round(stats.norm.ppf(1-alpha),2)
print("Z-score:", z_critical)

er=z_critical*(SD/math.sqrt(s1))
L=m_sample_1-er
H=m_sample_1+er

print("Confidence Level", L, H)
print("[",L,pm,H,"]")
```

```
URK21CS1128
Sample mean of 500 samples: 55.11994
Sample SD of 500 samples: 26.150292078605926
Z-score: 1.64
Confidence Level 53.20199835240902 57.03788164759098
```

[53.20199835240902 55.67213 57.03788164759098]

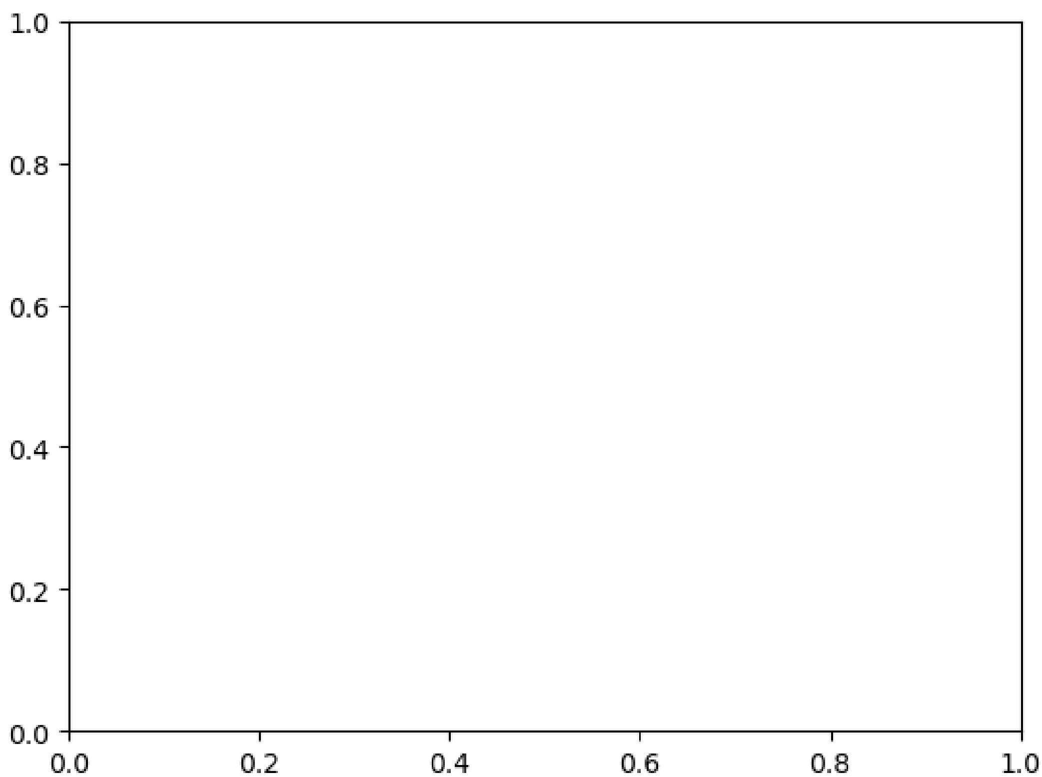
[35]:
```
#6. Calculate and plot the Confidence Intervals for 25 Trials with n=500 and␣
↪CI=95% for 'Unit price' column. Observe the results.
print('URK21CS1128')
sample_size = 500
intervals = []
sample_means = []

Cl = 0.95
alpha = (1-CL)/2
z_critical = round(stats.norm.ppf(1-alpha),2)

pm=df['Unit price'].mean()

#25 trials
for sample in range(25):
    sample = np.random.choice(a = df['Unit price'], size = sample_size)
    sample_mean = sample.mean()
    sample_means.append(sample_mean)

    sam_stdev = sample.std()
    margin_of_error = z_critical * (sam_stdev/math.sqrt(sample_size))
    confidence_interval = (sample_mean - margin_of_error,
                           sample_mean + margin_of_error)
    intervals.append(confidence_interval)
print(sample_means)
print(pm)
print(intervals)

plt.errorbar(x = np.arange(0.1, 25, 1),
             y = sample_means,
             yerr = [(top-bot)/2 for top,bot in intervals],
             fmt = 'o')

plt.hlines(xmin=0, xmax=25,
           y=pm,
           linewidth=2.0,
           color="red")
```

```
URK21CS1128
[56.262339999999995, 53.840920000000004, 55.37760000000001, 56.20538,
56.338860000000004, 55.26624, 55.78316, 57.87631999999999, 55.531180000000006,
56.09314, 55.38618, 54.94725999999999, 54.79228, 56.21382, 55.398300000000006,
56.891119999999994, 55.113839999999996, 53.622260000000004, 56.902440000000006,
56.5573, 54.9411, 53.122699999999995, 54.938840000000006, 54.75488000000001,
55.110699999999994]
55.67213
```

```
[(54.34959141456559, 58.1750885854344), (51.98789617172606, 55.69394382827395),
(53.408255464351384, 57.34694453564863), (54.277625356878865,
58.13313464312113), (54.44579980285873, 58.23192019714128), (53.36075274861317,
57.171727251386834), (53.840521208150435, 57.72579879184957),
(55.91333621616036, 59.839303783839625), (53.61959061399828,
57.442769386001736), (54.17653500254634, 58.00974499745366), (53.50070610493708,
57.271653895062926), (53.07762113601136, 56.81689886398863), (52.86297994712205,
56.72158005287795), (54.20915366202474, 58.21848633797526), (53.493806588321355,
57.30279341167866), (54.942762936979804, 58.83947706302018),
(53.133054827704086, 57.094625172295906), (51.70899530226765,
55.535524697732356), (54.98836946081282, 58.81651053918719),
(54.663622911404616, 58.45097708859538), (53.014528416474825,
56.86767158352517), (51.18324781631504, 55.06215218368495), (53.038358586861584,
56.83932141313843), (52.78397104250662, 56.72578895749339), (53.20330456133676,
57.01809543866323)]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[35], line 28
     25 print(pm)
     26 print(intervals)
---> 28 plt.errorbar(x = np.arange(0.1, 25, 1),
     29              y = sample_means,
     30              yerr = [(top-bot)/2 for top,bot in intervals],
     31              fmt = 'o')
     33 plt.hlines(xmin=0, xmax=25,
     34            y=pm,
     35            linewidth=2.0,
     36            color="red")

File /opt/anaconda3/lib/python3.9/site-packages/matplotlib/pyplot.py:2564, in␣
 ↪errorbar(x, y, yerr, xerr, fmt, ecolor, elinewidth, capsize, barsabove,␣
 ↪lolims, uplims, xlolims, xuplims, errorevery, capthick, data, **kwargs)
   2558 @_copy_docstring_and_deprecators(Axes.errorbar)
   2559 def errorbar(
   2560         x, y, yerr=None, xerr=None, fmt='', ecolor=None,
   2561         elinewidth=None, capsize=None, barsabove=False, lolims=False,
   2562         uplims=False, xlolims=False, xuplims=False, errorevery=1,
   2563         capthick=None, *, data=None, **kwargs):
-> 2564     return gca().errorbar(
   2565         x, y, yerr=yerr, xerr=xerr, fmt=fmt, ecolor=ecolor,
   2566         elinewidth=elinewidth, capsize=capsize, barsabove=barsabove,
   2567         lolims=lolims, uplims=uplims, xlolims=xlolims,
   2568         xuplims=xuplims, errorevery=errorevery, capthick=capthick,
   2569         **({"data": data} if data is not None else {}), **kwargs)

File /opt/anaconda3/lib/python3.9/site-packages/matplotlib/__init__.py:1442, in
 ↪_preprocess_data.<locals>.inner(ax, data, *args, **kwargs)
```

```
   1439 @functools.wraps(func)
   1440 def inner(ax, *args, data=None, **kwargs):
   1441     if data is None:
-> 1442         return func(ax, *map(sanitize_sequence, args), **kwargs)
   1444     bound = new_sig.bind(ax, *args, **kwargs)
   1445     auto_label = (bound.arguments.get(label_namer)
   1446                   or bound.kwargs.get(label_namer))

File /opt/anaconda3/lib/python3.9/site-packages/matplotlib/axes/_axes.py:3642,␣
 ↪in Axes.errorbar(self, x, y, yerr, xerr, fmt, ecolor, elinewidth, capsize,␣
 ↪barsabove, lolims, uplims, xlolims, xuplims, errorevery, capthick, **kwargs)
   3639 res = np.zeros(err.shape, dtype=bool)  # Default in case of nan
   3640 if np.any(np.less(err, -err, out=res, where=(err == err))):
   3641     # like err<0, but also works for timedelta and nan.
-> 3642     raise ValueError(
   3643         f"'{dep_axis}err' must not contain negative values")
   3644 # This is like
   3645 #     elow, ehigh = np.broadcast_to(...)
   3646 #     return dep - elow * ~lolims, dep + ehigh * ~uplims
   3647 # except that broadcast_to would strip units.
   3648 low, high = dep + np.row_stack([-(1 - lolims), 1 - uplims]) * err

ValueError: 'yerr' must not contain negative values
```



[33]:
```
# Print all column names in the DataFrame
print(df.columns)
```

```
Index(['Height', 'Score', 'Age'], dtype='object')
```

[ ]:
```
#7. Calculate the Correlation Coefficient using Pearson for the given table.
print("URK21CS1128")
from scipy.stats import pearsonr
from scipy.stats import spearmanr
import matplotlib.pyplot as plt
x=[17,15,19,17,21]
y=[150,154,169,172,175]
corr, _ = pearsonr(x,y)
print('Pearsons correlation: %.3f' % corr)
```

```
URK21CS1128
Pearsons correlation: 0.721
```

[ ]:
```
#8. Calculate the Correlation Coefficient using Spearman for the given table
print("URK21CS1128")
x=[17,15,19,17,21]
y=[150,154,169,172,175]
corr, _ = spearmanr(x,y)
print('Spearmans correlation: %.3f' % corr)
```

```
URK21CS1128
Spearmans correlation: 0.667
```

[ ]:
```
#9. Calculate the Covariance Matrix for the given data and analyse it
print("URK21CS1128")
import pandas as pd
df = pd.DataFrame(
    {'Height': [64, 66, 68, 69, 73],
     'Score':[580, 570, 590, 660, 600],
     'Age':[29, 33, 37, 46, 55]}
)

cov_matrix = df.cov()
print(cov_matrix)
```

```
URK21CS1128
        Height   Score     Age
Height   11.50    50.0   34.75
Score    50.00  1250.0  205.00
Age      34.75   205.0  110.00
```

[ ]:

```python
#10. Perform a hypothesis testing with Z-test A herd of 1,500 steer was fed a special high-protein grain for a month, has the standard deviation of weight gain for the entire herd was 7.1 and average weight gain per steer for the month was 5 pounds. By feeding the herd with special high-protein grain, it is claimed that the weight of the herd has increased. In order to test this claim, a random sample of 29 were weighed and had gained an average of 6.7 pounds. Can we support the claim at 5 % level?
print("URK21CS1128")
xbar=110
mu=100
n=50
sd=15
z=abs(((xbar-mu)/(sd/math.sqrt(n))))
if(z>1.96):
    print("Reject H0")
else:
    print("Accept H0")
print(z)
```

```
URK21CS1128
Reject H0
4.714045207910317
```

Result:

We have successful executed the program and the output is displayed in the each cell of the code.